CENTRE FOR
SYSTEMS
SCIENCE

Simon Fraser University, Burnaby, B.C., Canada V5A 1S6 / (604) 291-4704

92-05764/

Chart Parsing
Head -Driven Phrase Structure Grammar

Fred Popowich and Carl Vogel

CSS-IS TR 90-01

February 1990

Chart Parsing
Head-Driven Phrase Structure Grammar

Fred Popowich and Carl Vogel

CSS-IS TR 90-01
CMPT TR 90-01

February 1990

Centre for Systems Science
School of Computing Science
Simon Fraser University
Burnaby, British Columbia
CANADA V5A 1S6

Fred_Popowich@SFU.BITNET
vogel@cs.sfu.ca

Abstract

A Prolog chart parser for Head-Driven Phrase Structure Grammar (HPSG) is described. The HPSG formalism has undergone significant modification since Proudian and Pollard first introduced an HPSG chart parser in 1985. This paper describes the impact of these developments on chart parsing methodology. It also presents a critical evaluation of aspects of HPSG which emerges from our implementation.

1. Introduction

Head-Driven Phrase Structure Grammar (HPSG) is in the family of linguistic theories known as unification grammars (Shieber, 1986). Seiffert (1987) has presented a system for parsing unification grammars in general, while Proudian and Pollard (1985) have provided an implementation of HPSG in particular. But, HPSG has undergone quite a bit of development since the original implementation. The number of grammar rules has been reduced from sixteen (at least, this is the number included in Proudian and Pollard's 1985 implementation of the theory) to four (Pollard and Sag, 1987), and their theoretical foundations have been reformulated: no longer are they context free string rewrites augmented with unification of shallow feature structure matrices, now the rules are feature structures constrained through subsumption, which happen to have an abbreviation in the language of string rewriting. With this development of the theory in mind, it is fruitful to reconsider its implementation. Herein, we describe our Prolog chart parser for the 1987 version of HPSG. We find that the implementation process has afforded us a position to comment on necessary extensions to chart parsers of "general" unification grammars and to offer constructive criticism of HPSG as a specific unification grammar formalism.

2. HPSG

The fundamental unit of discourse in HPSG is the *sign*. Actually, in this paper we restrict our discussion to what we will call *t-signs*, theory signs, which are theoretical constructs articulated as attribute value matrices — attributes and restrictions on their values abstracted from linguistic objects called signs. It is not within the scope of this paper to consider the ontological status of signs (cf. Pollard and Sag, 1987, ch.1), nor the relationship between t-signs and signs (though certainly, HPSG is open to constructive criticism in regard to the abstractions, like the LEX feature, that it does make). See (Vogel and Popowich, forthcoming) for an extensive discussion of HPSG as a formal language for expressing linguistic theory based on a two dimensional multiple inheritance hierarchy of types. The structure of a t-sign is an aspect of a primitive hierarchy, and the operation of the theory is an aspect of its operative hierarchy.

2.1. T-Signs — Primitive Hierarchy

T-signs are recursively defined attribute value matrices (AVMs). Their structure is assumed to be acyclic. The attributes and values attempt to capture syntactic and semantic information which seem to constrain signs. For our discussion, we need consider only a portion of these features.

The top level attributes of t-signs are SYN, SEM, and PHON, which indicate syntax, semantics, and phonology. T-signs are partitioned into phrasal t-signs and lexical t-signs. The distinction is that phrasal t-signs also have a top level feature called DTRS which contains constituency information, while lexical t-signs do not have this feature. HPSG

uses the DTRS feature of phrasal t-signs to represent the constituent structure of a mother t-sign in terms of its head daughter (HEAD-DTR) and its complements (COMP-DTRS). There are other types of daughters which need not concern us here.

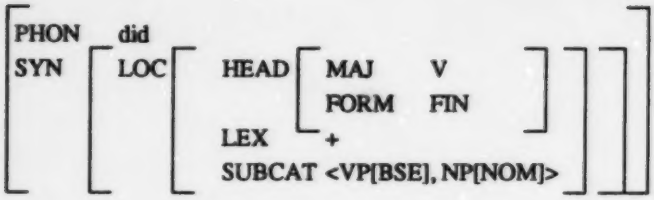
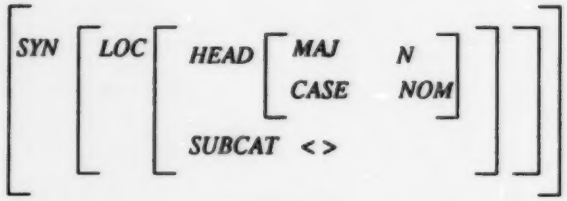


Figure 1: Lexical Entry for "did".

SYN values are classified into LOCAL and NONLOCAL features. Important LOCAL features are HEAD, SUBCAT and LEX. HEAD features record syntactic information usually associated with words, information like major category in linguistic classification, form (relative to category), case, aspects of agreement, and constraints on adjuncts. SUBCAT is a list of other t-signs, in decreasing order of obliqueness, with which the mother t-sign needs to combine in order to be saturated with respect to its classification in a subsumption hierarchy. LEX is a binary valued feature which correlates but does not coincide with the distinction between lexical and phrasal t-signs (Pollard and Sag, 1987, p.73).



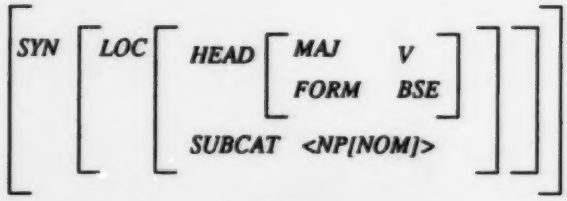


Figure 2: Sign Abbreviations.

Using these features, we may construct a t-sign to describe the lexical entry for "did" in Figure 1. Note that we have abbreviated this lexical sign by removing the NONLOCAL and the SEM attributes. The appearance of *VP[BSE]* and *NP[NOM]* in the

SUBCAT list indicates that "did" must combine with a base-form verb phrase and a nominative noun phrase to produce a "complete" constituent. *VP[BSE]* and *NP[NOM]* are actually abbreviations for the signs introduced in Figure 2 (Pollard and Sag, 1987, p.69).

2.2. Constraints on T-Signs — Operative Hierarchy

HPSG, as a linguistic theory, is in principle a discussion about constraints on the combination of t-signs. Its basic tenet is that the HEAD-DTR of a phrasal t-sign is itself a t-sign, and that syntactic and semantic information is shared between the two as well as among the other complements of the HEAD-DTR. The few grammar rules in the theory and the so-called universal and language specific principles mediate this sharing of information between heads and complement structures, actually, between the mother sign and its constituent daughters.

The grammatical principles independently constrain *information* contained in t-signs. The Head Feature Principle restricts the sharing of HEAD information between a mother t-sign and its HEAD-DTR. The Subcategorization Principle mediates SUBCAT information between a mother t-sign and its HEAD-DTR in terms of the mother t-sign's COMP-DTRS. The Semantics Principle defines the sharing of SEM information among a mother t-sign and all of its daughters, as does the English Constituent Ordering Principle for PHON information. The Adjuncts Principle, which plays a key role in the treatment of modifiers, actually violates the formal language used within HPSG since it is a relational constraint (Pollard and Sag, 1987, p.163). If the addition of relational constraints like those associated with the Adjuncts Principle can be shown equivalent to role value maps in KL-ONE, then the entire system may turn out undecidable (Schmidt-Schauβ, 1989). The intersection of these principles forms a major portion of the theory's operative hierarchy.

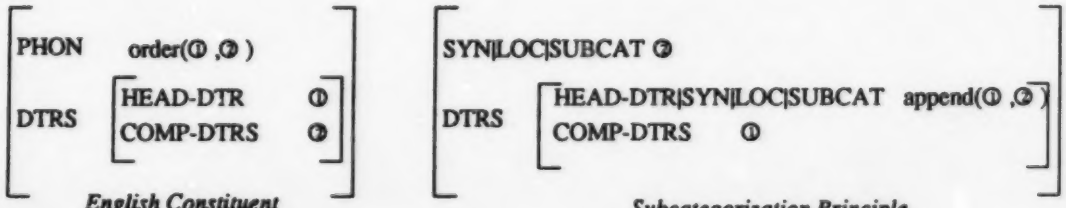
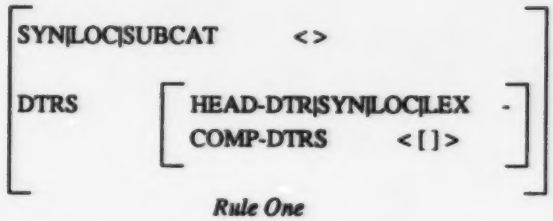


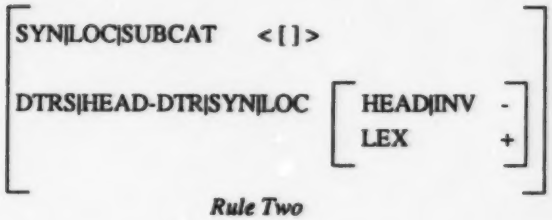
Figure 3: Subcategorization and Constituent Ordering Principles.

As the subcategorization and constituent ordering principles are essential to our discussion, they are illustrated in Figure 3. In these principles, re-entrancy (i.e., the sharing of a value by two features) is denoted by a common index. These signs follow the abbreviatory convention adopted by Pollard and Sag (1987) which allows us to reference values inside nested AVMs by writing the paths of attribute names (separated

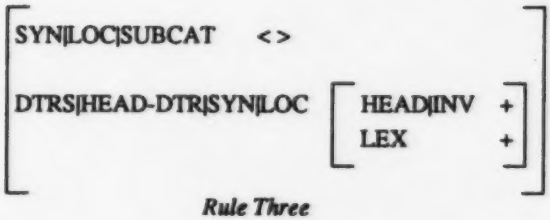
by vertical bars) which lead to these values. Our presentation of the principles as t-signs focuses on them as *nodes* in an inheritance network — the relative pseudocomplement operator ("=>") which is present in Pollard and Sag's formulation is *implicit* in the *links* of the operative hierarchy.



Rule One



Rule Two



Rule Three

Figure 4: Grammar Rules.

The "rules" of HPSG constrain the *structure* of constituency. The first rule is responsible for combining heads with their final complement (i.e., their "subject"). The second rule is for combining lexical heads with all but their final complements. The third rule coordinates lexical heads in inverted constructions where they combine with all of their complements including the subject. A fourth rule for treating adjuncts is tentatively put forward in (Pollard and Sag, 1987), but as is indicated (p.165), its necessity is uncertain. Due to the potential problems with the HPSG treatment of adjuncts, in this paper we have restricted ourselves to the first three grammar rules; these are shown in Figure 4. As each of the grammar rules is a t-sign in the operative hierarchy, it is also subject to the informational constraints imposed by the grammatical principles.

3. Chart Parsing HPSG

We have implemented a parser using modifications to the chart parsing paradigm due originally to Kay (1973) and Kaplan (1973), and more clearly described by Thompson (1981). The chart parser is written in Prolog, incorporating results from a similar system (Popowich, 1989a) for a related unification grammar formalism called Tree Unification Grammar (Popowich, 1989b). The general algorithm creates from input a table of well formed substrings in two steps: a predictor step finds grammar rules which match the category of the current substring and creates a new edge at that location with a list of expectations derived from the grammar rule (they predict what will combine next); a uniter step creates new edges by combining edges that have no expectations (they are inactive) with active edges that both i) meet the expectationless edge and ii) expect to combine with a substring of the same category as the expectationless edge. Seiffert (1987) extends Earley's chart-based recognition algorithm (Earley, 1970) to cover unification grammars by eliminating the addition of items to the chart which *subsume* elements that already exist. Earley's algorithm includes a redundancy check — it does not repeat the addition of an item if that item is already an element (i.e., the grammar is ambiguous). But, neither does the algorithm record the list of derivations all of which sanction the item as an element. Thus, it does not record all possible derivation trees for an item, only that it is derivable. Seiffert's algorithm extends this also by constructing an explicit representation of all parses during chart construction. Proudian and Pollard (1985) extend the method in a different way to permit processing in accord with the head-first, rather than left-first, requirements of HPSG, which include the fact that a head can meet its complement from the left or from the right.

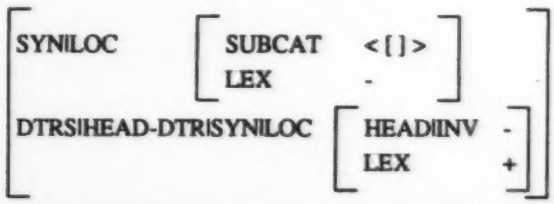


Figure 5: Revised Grammar Rule Two.

We find the subsumption test during a redundancy check on edges to be expensive, and we omit the check at the expense of space complexity. Like Seiffert, we maintain an explicit representation of parse trees during chart construction. Different parses of the same item will be stored in separate edges of the chart. Thus, all possible parse trees are retrievable. But if a grammar is left-recursive, a redundancy check on new edges is essential. For a head-driven unification grammar the equivalent to left-recursion is head-recursion, and HPSG actually turns out to be head-recursive in its specification of Rule Two, which does not specify that the mother t-sign unified with Rule Two is LEX - (and is therefore a phrasal t-sign). This means that Rule Two could be used again with the result of the previous unification as the head daughter, *ad infinitum*. However, since this

is the only instance of head-recursion in the grammar, we opted pragmatically to remove head-recursion from the grammar, rather than to incorporate the redundancy check into the parser. Our system uses a translation of the modified Rule Two given in Figure 5. Note that Rule Four, as stated in (Pollard and Sag, 1987), is also head-recursive, but also recall that its status within the theory is tenuous anyway: when articulated as an Adjuncts Principle, as constraints on the sharing of information rather than on constituent structure, the head-recursion is not introduced.

We also note that in the version of HPSG implemented by Proudian and Pollard, ordering information was encoded implicitly in the string rewriting notation of phrase structure rules. However, the most recent version of HPSG has an explicit constituent ordering principle which has been abstracted away from the rules. In this version of the theory the current expectation (complement) need not be adjacent to (i.e., meet) the head at all. This occurs in inverted constructions like "Did Mary walk?" in which the current expectation, corresponding to the most oblique complement using the ordering of the SUBCAT list, "walk," is separated from the head, "Did," by another word. Our approach, specific to the HPSG analysis of English, is to generate expectations at an edge from the SUBCAT list so that the current expectation is always textually adjacent to the head. That is, we create an expectations list which is the reverse of the subcategorization list. In the general case, the parser requires a less restricted definition of "meets" with additional constraint checking to verify the correspondence between analyzed substrings and the original string.

4. Implementation

Implementation of t-signs (called i-signs) and their incorporation into edges using the chart parsing paradigm are very separate issues. We have chosen an attribute value pair notation, implemented as Prolog lists, to construct i-signs which are very similar to t-signs. There are some notational differences between t-signs and i-signs; for instance, we specify that lexical i-signs have a null-valued *dtrs* attribute rather than no DTRS attribute at all. For some implementations of Prolog greater efficiency would obtain if we defined i-signs as fixed-arity terms instead of as lists.

Incorporation of i-signs into edges can proceed either explicitly, storing on any given edge the mother i-sign which contains all information present at the given level of analysis, or implicitly, where DTRS information is constructed after the parse in accordance with the rules and principles of HPSG. Note that this distinction between explicit and implicit representation is of a different order of granularity from the discussion in the previous section — there we indicated that constituent structure would be explicitly *represented*, here it is an issue of choosing between representation in the i-sign itself or in some fashion of encoding the constituent structure of the i-sign. While the latter approach is more space efficient, the former approach eliminates a step in processing (Seiffert, 1987). Our prototype system adopts the former approach. Thus, at

each edge we store the i-sign which corresponds in informativeness to the t-sign for the sentence at that particular level of analysis; this is the edge's "category".

As is clear from the presentation of t-signs for the various principles of HPSG, functions are included in the language to specify constraints on values. I-signs also allow this, even though Prolog does not support functions, simply by specifying a value as a variable to be instantiated when the relational constraints in which the variable participates are evaluated (Pereira and Shieber, 1987). These relations are recorded in edges as a list of pending restrictions as highlighted in Figure 6. When an inactive edge

```
edge([ [dtrs, [ [head_dtr, ISIGN0],
                [comp_dtrs, LIST_OF_ISIGNS]
                ...]],
  [phon, VALUE],...],
CONSTITUENT_LIST,
EXPECTATIONS_LIST,
[order(ISIGN0, LIST_OF_ISIGNS, VALUE), ...],
LEFT_POS,
RIGHT_POS,
EDGE_ID).
```

Figure 6: An Active Edge with a non-Empty List of Relational Restrictions.

is newly created through the uniter or the predictor, the restrictions that have accumulated from its constituent edges are evaluated, and the new edge is asserted with value variables instantiated, as if a function had been applied, and with a resultingly null list of unevaluated restrictions.

An exception is the implementation of the "append" function over SUBCAT lists from the subcategorization principle. The values of *subcat* on i-signs are structured as difference lists with tail variables. The value of a difference list of the form A\B is the list of elements that remain when the list of elements in B is deleted from the list of elements in A. Part of the power in the notation is in that the tail component can be given in terms of unbound variables which when instantiated correspond to elements which are effectively subtracted from the list. Thus, (using capitals for variables and lowercase letters for constants), the difference list [a,b,c|A]\[c|A] is equivalent to the list [a,b] in usual notation. Using difference lists allows us to efficiently partition a list with a single Prolog operation (Pereira and Shieber, 1987). For example, assume we want to remove an initial sublist [a,b] of some list [a,b,c] (as we need to do in the Subcategorization Principle) to yield a resulting list [c]. Since [a,b,c] is encoded as [a,b,c|A]\A, we need only unify this difference list with [a,b|B]\C which will result in binding A to C and B to [c|A]. We can then define the resulting list in terms of A and B (i.e., it is B\A, which is equal to [c|A]\A). This property of difference lists allows us to encode the Subcategorization Principle without the use of the "append" function, as illustrated in Figure 7. The list represented by the difference list *Comps\Tail* will contain all of the elements described by *Comps\Rest* plus those from *Rest\Tail*; the SUBCAT list of the

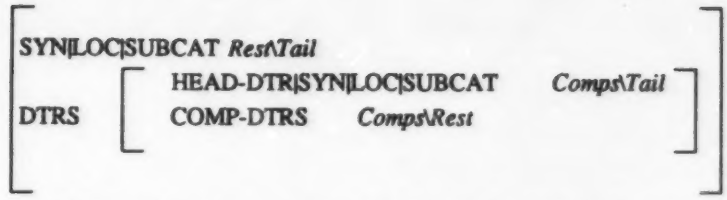


Figure 7: Subcategorization Principle with Difference Lists.

head daughter contains the COMP-DTRS followed by the elements from the SUBCAT list of the mother sign.

5. Example

Consider the following analysis of the question "Did Mary walk?" We begin by initializing the waiting list with three edges (edges pass through the waiting list before being added to the chart (cf. Thompson 1981)). The category of each corresponds to the i-sign associated with the lexical entry for each word in the sentence. None of the initial edges have expectations associated with them. For the mode of processing we use in the prototype (bottom-up parsing, last-in-first-out storage), the first spanning edge placed on the waiting list will cause termination when it is added to the chart.

A portion of the output from our system has been included in Figure 8 to demonstrate the sequence of edges considered. It indicates information contained at edges as they are constructed and placed on the waiting list for processing (not as they are actually inserted into the chart). The code which produces this output has been included in the appendix.

```
| ?- parse.
|: did mary walk?
initializing:  place inactive edge 1 ([did]) in waitinglist.
initializing:  place inactive edge 2 ([mary]) in waitinglist.
initializing:  place inactive edge 3 ([walk]) in waitinglist.
predictor:  place inactive edge 4 ([walk]) in waitinglist (used rule 2).
predictor:  place active edge 5, built from [did] in waitinglist (used rule 3).
uniter:  place active edge 6, built from [did] in waitinglist.
predictor:  place active edge 7, built from [walk] in waitinglist (used rule 1).
uniter:  place inactive edge 8 ([mary,walk]) in waitinglist.
uniter:  place inactive edge 9 ([did,mary,walk]) in waitinglist.
uniter:  place inactive edge 10 ([did,mary,walk]) in waitinglist.
```

Figure 8: Some Output from the Chart Parser.

The SUBCAT list of the i-sign for the input "did" is used in conjunction with Rule Three during the predictor step to create a new edge with expectations (it is thus an active edge) corresponding to the subject and VP complement of "did". The expectations list is in order of increasing obliqueness as explained earlier. Constraints associated with the various principles (i.e., constituent ordering) carried in the i-sign for Rule Three are annotated on the new edge. Constraints accumulate as edges combine and are evaluated only when combination results in an inactive edge.

During the uniter step, this new edge will meet the inactive edge associated with "mary." The i-sign category of the inactive edge will unify with the least oblique expectation from the active edge, resulting in a new, combined edge which contains the remaining expectations. Note that if the expectations list corresponded in order directly to SUBCAT, the subject which is the least oblique complement, would be the last expectation, and without a generalization to the definition of "meets," the two edges would not meet. The new active edge which still has the expectation for a VP complement will later meet the inactive edge associated with "walk." The i-sign of the inactive edge will unify with this remaining expectation to yield an inactive edge that spans the entire sentence. That the accumulated relational restrictions were met is implicit in the addition of the new edge to the waiting list. The new edge will have a null restrictions list.

Actually, because of a slight ambiguity in the grammar that we have implemented there will be two spanning edges on this sentence. These correspond to constituent analyses of the attachment of "walk" in which the LEX feature varies over both its possible values. This happens because, in general, we use SUBCAT as a list of t-signs without specifying whether they are lexical or phrasal. So, the simplistic analysis which combines "walk" as a lexical t-sign (LEX +) can be eliminated by specifying the verb phrase complement in the SUBCAT list to be LEX -. The phrasal (LEX -) analysis involves another edge which is created from the LEX + edge during the predictor step, in conjunction with Rule Two applied to a lexical head having no complements.

Other edges will also be constructed, although they do not ultimately combine into a spanning edge. For instance, an analysis of the substring "mary walk" will be encoded in an edge since further processing of the LEX - edge for "walk," mentioned above, will create another active edge in conjunction with Rule One during the predictor step. This edge will record the expectation of a least oblique complement which will be met by the edge corresponding to "mary."

6. Discussion

The process of constructing a machine implementation of HPSG has provided us with an excellent position from which to assess the generality of chart parsing algorithms and has also allowed us to provide constructive criticism of HPSG.

We began by distilling HPSG into a formal language of t-signs, and we constructed a formally equivalent notation of i-signs in Prolog. In processing these, we found some ambiguities in the HPSG grammar, most of them related to the specification of the LEX feature. We also found aspects of the theory (i.e., the Adjuncts Principle) which call into question its otherwise attractive computational complexity. We proposed a general method for dealing with head-recursion in unification grammars, arguing that removing it actually is general given the massive lexicalization (and minimal rule specification)

inherent in many of these formalisms.

Our ongoing work involves further investigation of HPSG as a formal language and encoding a larger portion of the HPSG grammar. We intend to include an additional rule and an associated principle (Pollard and Sag, forthcoming) which together cover topicalization. Finally, we are also examining ways to include some version of the Adjuncts Principle and the Semantics Principle. We expect further insights into chart parsing methodologies to emerge from this work, as well as the production of a useful tool for examining unification grammars in general.

Acknowledgements

We would like to thank Dan Fass for his helpful comments on earlier drafts of this paper. We are also grateful to Mark Gawron for numerous discussions about much of the material contained herein. The research presented in this paper is supported by the Advanced Systems Institute of British Columbia, and by the Natural Sciences and Engineering Research Council of Canada under Grant Nos. OGP0041910 and A4309.

References

Earley, Jay. (1970). An Efficient Context-Free Parsing Algorithm. *Communications of the ACM, 13*, 94-102.

Kaplan, Ron. (1973). A General Syntactic Processor. In R. Rustin (Ed.), *Natural Language Processing*. Algorithmics Press, New York, NY.

Kay, Martin. (1973). The MIND System. In R. Rustin (Ed.), *Natural Language Processing*. Algorithmics Press, New York, NY.

Pereira, Fernando, and Stuart Shieber. (1987). *Prolog and Natural-Language Analysis*. Centre for the Study of Language and Information, Stanford University, CA.

Pollard, Carl, and Ivan Sag. (1987). *Information-Based Syntax and Semantics, Volume 1: Fundamentals*. Centre for the Study of Language and Information, Stanford University, CA.

Pollard, Carl, and Ivan Sag. (forthcoming). *Information-Based Syntax and Semantics, Volume 2: Topics in Binding and Control*. Centre for the Study of Language and Information, Stanford University, CA.

Popowich, Fred. (1989). *A Tree Unification Grammar-Based Natural Language Processor* (CSS-IS TR 89-08). Simon Fraser University, Burnaby, B.C.

Popowich, Fred. (1989). Tree Unification Grammar. In: *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*. Vancouver, B.C.

Proudian, Derek and Carl Pollard. (1985). Parsing Head-Driven Phrase Structure Grammar. In: *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*. University of Chicago, Chicago, IL.

Schmidt-Schauβ, Manfred. (1989). Subsumption in KL-ONE Is Undecidable. In: *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*. Toronto, Ontario.

Seiffert, Roland. (1987). *Chart-Parsing of Unification-Based Grammars with ID/LP-Rules* (LILOG-REPORT 22). IBM Deutschland GmbH, LILOG, Stuttgart, West Germany.

Shieber, Stuart. (1986). *An Introduction to Unification-Based Approaches to Grammar*. The University of Chicago Press, Chicago, IL.

Thompson, Henry. (1981). Chart Parsing and Rule Schemata in PSG. In: *Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics*. Stanford University, Stanford, CA.

Vogel, Carl and Fred Popowich. (forthcoming). *Head-Driven Phrase Structure Grammar as an Instance of Inheritance Based Reasoning* (CSS/LCCR TR 90-03, CMPT TR 90-03). Simon Fraser University, Burnaby, B.C.

A. Prolog Code

The following code is taken from our Prolog HPSG parsing system. Only the primary predicate definitions are included. The definitions of the utility predicates are relatively straightforward, and the behaviour of these predicates should be apparent from their names (i.e., *clear_edges*, *set_num_to_one*). Their presentation is inessential to a discussion of the chart parser.

```
%% The Parser

parse :-
  clear_edges,
  set_num_to_one,
  read_sent(WordList),
  init_chart(WordList),
  select_next(Edge),
  parse(Edge,WordList).

init_chart([]).

%Find all lexical entries corresponding to this word.
%Note conversion to difflist notation on the value of PHON.

init_chart([Word|Sentence]) :-
  entry([_dtrs,[phon, [Word]]|Sign]),
  make_edge([_dtrs,[phon, [Word|E]\E]|Sign],
        [],
        [],
        [],
        [Word|Sentence],
        Sentence),
  output(initializing,[_dtrs,[phon, [Word|E]\E]|Sign],waitinglist,[],Rule),
  fail.                                          % look for other entries


%For robustness, if the word is unknown to our lexicon, assume that
%it is a word, subsequent analysis will give us more information about
%the word via unification.
%Again, note conversion to difflist notation on the value of PHON.

init_chart([Word|Sentence]) :-
  \+ entry([_dtrs,[phon, [Word]]|Sign]),  % no entry for word
  sign(word,[_dtrs,[phon, [Word]]|Sign]), % create an "empty" lexical isign
  make_edge([_dtrs,[phon, [Word|E]\E]|Sign],
        [],
        [],
        [],
        [Word|Sentence],
        Sentence),
  output(completer,ActSign,waitinglist,Rest,Rule),
  fail.

init_chart([Word|Sentence]) :- init_chart(Sentence), !.


% Edge format:
%   edge(Category, Constituents, Expects, FullSign, LeftPos, RightPos, Num).

make_edge(Category, Constituents, Expects, FullSign, LeftPos, RightPos) :-
  numgen(N),
  recorda(
    to_process,
    edge(Category, Constituents, Expects, FullSign, LeftPos, RightPos, N),
    _).
```

```
%This "parse" predicate describes a successful parse.  If it succeeds then
%an edge has been constructed accounting for the information in the entire
%sentence.  View it as a constructive proof:  ISign is the construction.

parse(edge(ISign,Constituents,[],Parse,Wordlist,[],Num),Wordlist) :-
  path(ISign,phon,Phonology),
  undiff(Phonology,Wordlist),
  draw_sign(ISign,0).


%NOTE:  Let A mean, "an active edge", and I, "an inactive edge"

%This "parse" predicate encodes the essential behavior of the chart parser.

parse(Edge,Wordlist) :-
  fundamental_rule(Edge),       %See if A->I or I->A with Edge.
  find_rules(Edge),             %Apply directional strategies.
  fail.                         %Try other alternatives


% An inactive edge meets an active edge:

fundamental_rule(edge(PSign,Const,[],[],Lpos,Rpos,Num)) :-
  edge(ActSign,ActConst,[Exp|Rest],Constraints,ActLpos,ActRpos,_),
  meet(Lpos,Rpos,ActLpos,ActRpos,NewLpos,NewRpos),
  sign_unify(Exp,PSign),
  process(Constraints,Constraints1,Rest),
  make_edge(
         ActSign,
         [[Num,frne]|ActConst],
         Rest,
         Constraints1,
         NewLpos,
         NewRpos),
  output(completer,ActSign,waitinglist,Rest,Rule),
  fail.


% An active edge meets an inactive edge:

fundamental_rule(
            edge(ActSign,ActConst,[Exp|Rest],Constraints,ActLpos,ActRpos,_)  ):-
  edge(PSign,Const,[],[],Lpos,Rpos,Num),
  meet(Lpos,Rpos,ActLpos,ActRpos,NewLpos,NewRpos),
  sign_unify(Exp,PSign),
  process(Constraints,Constraints1,Rest),
  make_edge(
         ActSign,
         [[Num,frle]|ActConst],
         Rest,
         Constraints1,
         NewLpos,
         NewRpos),
  output(completer,ActSign,waitinglist,Rest,Rule),
  fail.

fundamental_rule(_).

meet(P1, P, P, P2, P1, P2) :- !.
meet(P, P2, P1, P, P1, P2).

%Triggered on inactive edges.

find_rules(edge(PSign,Const,[],Parse,Lpos,Rpos,Num)) :-
  get_rule(R, Rule, Constraints),
  path(Rule, dtrs:head_dtr, HSign),     %get the value at dtr:head_dtr
  sign_unify(HSign,PSign),              %see if values unify
  path(Rule, dtrs:comp_dtrs,Daughters), %get the value of comp_dtrs
```

```
    undiff_exp(Daughters,Expects),          %convert Difflist to expectations
    process(Constraints,NConstraints,Expects),
    make_edge(Rule, [[Num,R]], Expects, NConstraints, Lpos, Rpos),
    output(predicter,Rule,waitinglist,Expects,R),
    fail.

find_rules(_).

% don't process constraints if there are expectations

process(C, C, [_|_]) :- !.
process([], [], _).
process([C|Rest], Result, Es) :- call(C), process(Rest, Result, Es).

% The Constituent Order Principle, though stated declaratively
% as a t-sign, references a function of t-sign and lists of t-signs
% to phonological values.

% order(Sign, Difference List, Result)

order(Head_Dtr, Comps\T, Phonology) :-   %[Head|Result]\R) :-
  path(Head_Dtr, syn:loc:lex, plus), !,  % lexical head
  path(Head_Dtr, phon, Head),
  order(Comps\T, Result\R),
  concat_dl(Head,Result\R,Phonology).

order(Head_Dtr, Comps\T, Result\R) :-    % nonlexical heads = most oblique comp
  order([Head_Dtr|Comps]\T, Result\R).

order(Comps, Result\Result) :-
  empty_dl(Comps),!.                     % For complements

order([Sign|Comps]\T, Phonology) :-
  path(Sign, phon, Phon),
  order(Comps\T, Result\R),
  concat_dl(Result\R,Phon,Phonology).

get_rule(N,Matrix,Constraints) :-
  rule(N,PSign),
  eval_functions(PSign,Matrix,Constraints).

% Since this is all supposed to be declarative, this relation exists
% solely to declare the relation of a sign created by unification with
% a rule (hence, it contains further statements of relationships like
% 'concat') and and the sign which contains (instead of the function
% names) the value implied by the relationships stated in the first
% sign.  (Procedurally:  we render a sign which has all its functions
% evaluated.)

eval_functions([[dtrs|D], [phon,order(A,B)] | Rest],
               [[dtrs|D], [phon,C] | Rest],
               [order(A,B,C)]) :- !.

eval_functions(M, M, []) :- !.

% path(Sign,Path,Value)

% For feature value pairs

path([Feature,Value],Feature:Rest,Return) :- !, path(Value,Rest,Return).
path([Feature,Value],Feature,Value) :- !.

% we have to contend with a list of avms

path([[F,V]|_],Path,Return) :- path([F,V],Path,Return), !.
path([[_,_]|OtherValues],Path,Return) :- path(OtherValues,Path,Return).
```

Centre for Systems Science

• Simon Fraser University established the **Centre for Systems Science** as a provincial centre for advanced study. The objective of CSS is twofold: coordinate and combine advanced systems research at SFU, and encourage application of research results within industry in order to further the technological development of British Columbia.

• CSS has assembled three research groups to streamline its efforts. The orientation of these groups is as follows: intelligent systems, microelectronics, and computer and communications systems. Because of shared research interests, this latter group merges with LCCR (**Laboratory for Computer and Communications Research**) for the publication of technical reports.

• CSS's three groups involve over 80 skilled faculty researchers principally from SFU's Schools of Engineering Science and Computing Science, but also draw substantially on Mathematics, Physics, Geography, Philosophy, Linguistics, Psychology, Communication, and Kinesiology.